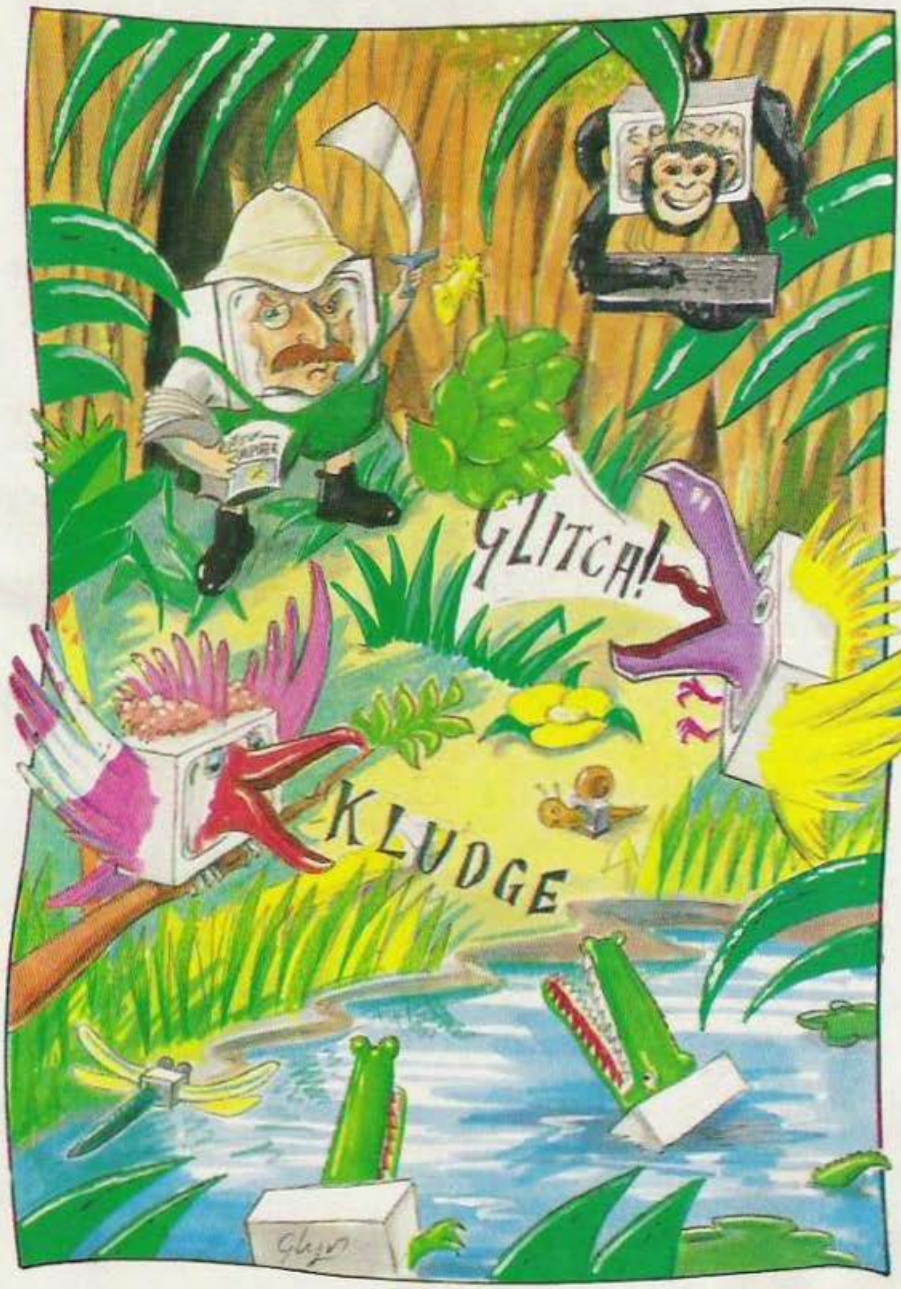
Your COMPUTER COURSE

JULY 1986

A COMPREHENSIVE GUIDE TO PROGRAMMING – PART 5



HACKING THROUGH THE JARGON JUNGLE

OUR GLOSSARY SHOWS YOU THE WAY

SPEAKING THE BBC LANGUAGE

AN INTRODUCTION TO 6502 MACHINE CODE PROGRAMMING

GETTING TO GRIPS WITH GRAPHICS

MOVING IMAGES ON-SCREEN

FORCED TO SUBMIT

THE CP/M SUB-COMMAND EXPLAINED

The short but intensive *Your Computer* Course is almost at an end. This month's instalment is the penultimate part of the six-monthly series designed to introduce the fundamentals of computer programming.

The mix of articles this month follows that established in earlier parts of the course, with a range of subjects encompassing all aspects of the programming art.

We begin with further exploration of the techniques of computer graphics. In particular, the Basic UDG program for the CBM64, as published last month, is modified to increase its flexibility.

The *Your Computer* Course glossary has been a major feature in previous parts and this month is no exception. John Lettice nears the end of the alphabet; it is surprising how many computing terms begin with the letters from Q to S.

The third of this month's articles introduces 6502 machine coding programming. While not one of the latest 16-bit wonder MPUs, the 6502 is the workhorse at the centre of many leading micros, including the BBC computer.

Finally this month we return to the subject of computer operating systems with a detailed look at the powerful CP/M submit utility.

INDEX

Computer Graphics 34
Glossary (Q-S) 36
6502 programming 38
CP/M Submit 40

Contributors:
Ken Alexander
John Lettice
Peter Turcan
Geof Wheelwright

Computer-animated design

Geof Wheelwright explores putting sureal graphics into motion.

Animation is the cornerstone of all good arcade games on home micros, giving the player the feeling that they are really there in whatever fantasy-scape the game planner has designed. To round off the taster on animation techniques you will need to write your games, we will run over the program example we gave you last time, giving you the information you need to finish the rock-collecting game and suggesting improvements you might make on it.

The first thing you need to resolve is the question of smooth movement across the screen. As you will know by now, the simplest method of moving something across the screen is to use the PRINT statement to place the character on the screen; then to use the CLS statement to clear the screen of what you have printed, and then to use a PRINT statement to print the character one place further than last time.

That method of animation works but is both slow and clumsy. It also causes a good deal of screen flicker as the whole of the screen is being cleared every time you want to move the character, not just the space where the character had previously been printed.

To make your animation truly smooth, you need to arrange things so that when you move a character from one spot to another – whether it is a number, letter or part of a user-defined graphic – the place it has just come from is clear. As you will want to move your character up and down as well as back and forth, that means ensuring that the business of blanking-out the previous position of the character will have to be handled in two dimensions, not just one.

The exact method by which you set about identifying the two-dimensional position of a character is different from machine to machine. Many just use x and y co-ordinates to allow you to specify the exact position of a character of pixel, while others map the screen from the top left-hand corner to the bottom right-hand corner.

On computers which use the latter technique such as the Commodore 64 and 128, you will need to determine the exact single-number screen locations which correspond to where you want to print your character and where you want to print a blank space in place of where it had been. On those which use the x,y placement, you need only subtract or add to the value of x and y, depending on where in the screen to which you have moved the character.

Once you have mastered this concept, it is a relatively short leap to achieving full cursor control of the character. All you have to do is have the value of the 'Y' co-ordinate to increase when you press the up key or move the joystick upwards – and conversely get it to decrease when you move the joystick down or press the down key – and tie the X value to the press of the left and right keys.

In the Commodore 64 program example, you can see how the variable 'Q' is used to define the position of the character in our rock-collecting game.

Only one variable is needed because the screen positions of characters and pixels on the Commodore 64 are not of the X,Y type. They are at the top left-hand corner of the screen with one number, and increase until you reach to the bottom left-hand corner of the display.

Recognition must be achieved

For the necessary cursor control to play the game, you see how we need to take into account what happens if each of the four up, down, left and right keys (CHR$(145),CHR$(17), CHR$(157) and CHR$(29)) are pressed. You will also see that because the on-screen man is made up of not one, but four,

special characters that each of the characters has to be rePOKED to the screen every time the man is moved.

The other major game component you will notice is the counter in line 3000. At the moment, it is only really set up as a timer, giving you only 125 presses of the keyboard to complete all collection of the on-screen rocks. You could make this a count instead by typing:
FOR T=125 to STEP –1
at line 3000 instead, which would start the countdown of your kepresses at 125 and move down to 1.

The alternative, is to score only when a rock is collected. To do that, however, you would need to write a routine which found where each of the randomly-placed on-screen rocks finished and match that constantly against your current position. When it detected a match – or what we in the game business call a collision – you would score and the place which had contained the rock would be blanked-out.

Modularity and structure

To make the game more sophisticated, you could then draw a series of more complex backgrounds in which the hero could move and then perhaps some animation sequences – i.e., walking or picking up a rock – which could be invoked at certain points in the program.

Remember that the most important thing in any program is modularity and structure, so that as you move round and improve and change one piece of the program the next is not affected. In the rock-collecting program, for example, the keyboard input section is all one module and could be replaced by a joystick input section.

To allow for joystick input, read the values of the joystick port instead of the keyboard and use the same incrementation for when the cursor is moved to gain the necessary control to play the game.

Finally, to our little secret about how to solve that little problem in the game you saw last time. I will not tell you at once where the change is but suffice it to say that you will see it between lines 10 and 230.

```
 4 PRINT CHR$(147)
10 POKE 56334, PEEK (56334) AND 254: POKE 1,PEEK(1)AND251
20 FOR I=0 TO 63
30 FOR J=0 TO 7
40 POKE12288+I*8+J,PEEK(53248+I*8+J)
50 NEXT J:NEXT I
60 POKE1,PEEK(1)OR4:POKE56334,PEEK(56334)OR1
70 POKE 53272,(PEEK(53272)AND240)+12
80 FOR CHAR=60TO63
90 FOR BYTE=0TO7
100 READ NUMBER
110 POKE 12288+(8*CHAR)+BYTE,NUMBER
120 NEXT BYTE:NEXT CHAR
130 FOR L=40 TO 78
140 PRINT CHR$(147)TAB(1)CHR$(60);
150 PRINT CHR$(61)TAB(L)CHR$(62)CHR$(63)
160 FOR H=1 TO 10:NEXT H:NEXT L
200 DATA 7,7,7,1,1,31,3,3
210 DATA 224,224,224,128,128,248,192,192
220 DATA 1,2,4,8,16,32,64,128
230 DATA 128,64,32,16,8,4,2,1
1000 PRINT CHR$(147)
1500 PRINT CHR$(5)
2000 Q=1484
2100 FOR I=55296 TO 56295: POKEI,1:NEXT I
2600 FOR U=1 TO 12:X=INT(RND(1)*1000)+1024:POKE X,35:NEXT U
3000 FOR T=1 TO 125
3100 GET B$:IF B$="" THEN 3100
4000 IF B$=CHR$(17)THEN Q=Q+40:POKE Q-40,32:POKE Q-39,32:POKE
Q,32:POKE Q+1,32
5000 IF B$=CHR$(145)THEN POKE Q,32:POKE Q+1,32:POKE
Q+40,32:POKEQ+41,32:Q=Q-40
6000 IF B$=CHR$(157)THEN POKE Q,32:POKE Q+1,32:POKE
Q+40,32:POKEQ+41,32:Q=Q-1
7000 IF B$=CHR$(29)THEN POKE Q,32: POKE Q+1,32:POKE
Q+40,32:POKEQ+41,32:Q=Q+1
8000 POKE Q,60:POKE Q+1,61
8200 POKE Q+40,62:POKE Q+41,63
9000 PRINT T:PRINT CHR$(19):PRINT CHR$(32):NEXT T
9500 PRINT CHR$(147)
9700 POKE 53272,21
10000 FOR G=1 TO 23
11000 PRINT "...............
TIME'S UP....."
12000 NEXT G:GOTO 10000
```

Your COMPUTE

The jargon which pervades the comp language than a tool to effective com know. By following John Lettice can be fluent in the hi-tech pa

Q

■ **Qwerty:** Used by manufacturers in advertising blurb for their computers, as in "has full qwerty keyboard". Beware of non-qwerty keyboards.

■ **Qdos:** DOS stands for disc operating system, therefore Qdos is the operating system for the Sinclair QL, launched without discs. In this case the D obviously has to stand for something else, but please don't write.

■ **QL:** Machine launched in 1984 by Sinclair Research, reputedly with more operating system versions than machines sold. Sir Clive said it stood for Quantum Leap; the full name of the machine is now known to be Quiche Lorraine.

■ **Quinkey:** Five-key single-handed word processing system used on the Microwriter. Beware of non-qwerty keyboards – see qwerty.

■ **Quadrant:** Part of Sutton where computer journalists rub shoulders with esoteric titles dealing with supermarkets, travel agents and other publications.

■ **Queue:** What you had to do initially if you wanted to buy a QL. Also refers to an ordering of data awaiting processing.

R

■ **RAM:** Random Access Memory. Not, as you might think, a medical condition involving forgetfulness but a kind of memory used in computers which allows you to access any part of it, from any point, at any time.

■ **ROM:** Read Only Memory. The microcomputing equivalent of the Bourbons, who learned nothing and forgot nothing. ROM is a kind of memory which can be read but cannot be erased and so cannot be changed. usually used to contain the "personality" of a computer.

■ **REM:** Short for remarks and used by programmers to annotate programs with comments explaining a listing. Computers ignore those unco-ordinated and misleading burblings just as you, if confronted by a programmer, would do.

■ **Random access:** The entryphone's broken. Also used of disc drives, which generally allow the read head to jump to any part of a disc at any time, as opposed to tape drives which, because they have to read through all matter preceding the required entry, use sequential, or serial, access.

■ **Raster:** The little dot which scans across the TV screen to make up the picture. First rose to prominence in Bob Marley's seminal TV repair concept album "Raster Scan Vibrations."

■ **RUN:** 1, Basic command used to execute a program after is has been written. 2, Basic action taken by programmers to avoid execution after a program has not been written.

■ **Real number:** The profit figures of a normal computer company, as opposed to the profit figures of IBM, which are absolutely unreal. Also all sensible numbers which people can comprehend, consisting of positive, negative, zero and fractions.

■ **Real-time clock:** A software device, sometimes built into a computer ROM, which gives you progressively more misleading information about the time of day the longer the computer is turned on.

■ **Reciprocal:** The reciprocal of a number is one divided by that number and relates in the same way to that number as a normal computer company's profits relate to those of IBM.

■ **Record:** The number of operating system versions the QL has had. Alternatively, a collection of data items or a subdivision of a file.

■ **Recursion:** The repetition of a series of operations, which may include an operation that calls itself. See **bug**.

■ **Redundant:** Duplication of functions; description of a function no longer needed; what dictionary authors are when they reach Z.

■ **Refresh:** To re-write data into an area, often used describing the updating of a screen.

■ **Register:** What you should have done by May 11 if you keep data on people and you do not want to be nicked. Also a temporary storage area within memory.

■ **Re-locatable:** A program which can be placed in different areas of memory is re-locatable. So are other things. If price and quality of current manufacturers of Sinclair machines does not suit Alan Sugar, production is re-locatable to Korea.

■ **Report generator:** Usually part of another program, e.g., an accounting suite, which produces printed output.

■ **Reserved word:** A word which cannot be used as a variable, often because it is a Basic command.

■ **Re-set:** What you have to do when your program crashes. There are three kinds – soft re-set, which re-sets the machine through software; hard re-set, which involves switching it off; and very hard re-set, the hammer which returns the machine to its pre-manufactured state.

■ **Return:** Computer key used to execute Basic commands, sometimes also known as Enter.

■ **Right-justified:** Speech by Norman Tebbit, or hard copy where text is aligned on the far right. As I said, speech by Norman Tebbit.

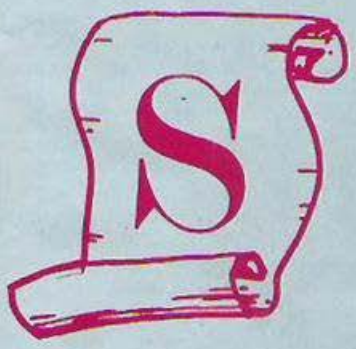
■ **Round:** Rounding is the process of chopping-off the twiddly bits at the end of long decimal numbers, thus causing a rounding error.

■ **Routine:** Short for subroutine.

■ **RS232:** A standard interface designed to facilitate communications. It is important to make sure you have the correct standard, as they are all different.

R COURSE

er industry seems more like a foreign unications for those who are not in the guide to computer-speak you, too, ance spoken by computer people.



■ **Scrolling:** Any text moving up the screen faster than you can read comfortably is scrolling.

■ **Search:** A search checks data for the presence of a given value.

■ **Segment:** Computer component – see **re-set.** Alternatively a division of memory.

■ **Semiconductor:** London Transport trainee, or an alternative expression for an integrated circuit.

■ **Sequencer:** A device which can arrange data in a particular order, i.e., a London Transport signalman.

■ **Serial:** A kind of data transmission which involves the elements to be sent lining-up conga fashion, rather than parallel, which involves them holding hands and jumping.

■ **Shared resources:** Items of hardware which can be used by more than one micro, pioneered unsuccessfully by Widgit Data Products with its One Per Building concept.

■ **Shift:** A key which alters the case or mode of a micro, or to move data in a register to the left or to the right.

■ **Signal:** An electrical impulse carrying information between machines.

■ **Sign-on:** Action followed to gain access to a remote computer, synonomous with log-on. Also see **redundant.**

■ **Simulator:** A device, also known as an emulator, which simulates another device, generally not very accurately or quickly.

■ **Sinclair:** Term used to describe mail order problems. Now obsolete.

■ **Sink:** What happens to micro makers after a bad Christmas. A method of dissipating heat generated by circuitry.

■ **Source code:** The language in which a program is written initially.

■ **Spooling:** Involves writing data to a disc for later output to a printer.

■ **Start bit:** A bit preceding the data bits in a character code during communications to identify the first data bit. A stop bit identifies the last one.

■ **String:** A series of characters.

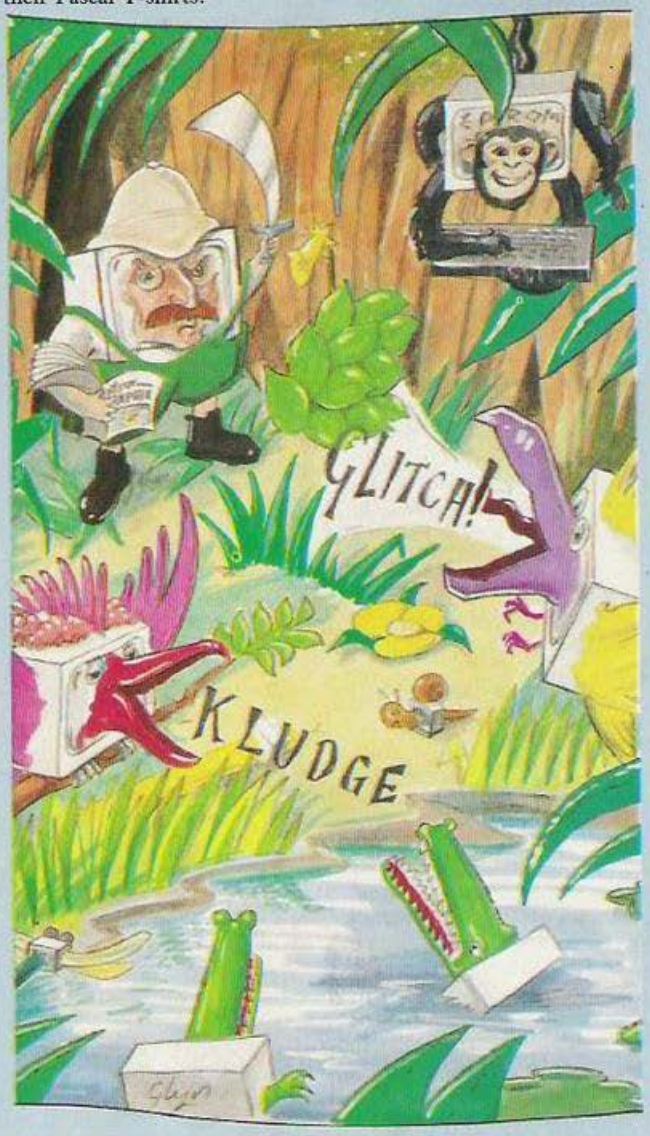
■ **Structured programming:** A method of prgramming based on neat flowcharting and intended to be elegant – good structured programmer word, this – and easy to understand. Structured programmers can be recognised by their willingness to discourse on esoteric program languages and occasionally by their Pascal T-shirts.

■ **Subscript:** Sometimes refers to illegible printer output, or rejected BBC radio plays, but really means those dinky little letters which reside just below the real thing.

■ **Subset:** A collection of related data which is part of, and completely encompassed by, a larger collection of data, or set.

■ **Syntax error:** A typical piece of computer output.

■ **Systems analysis:** A method used by consultants to persuade you to pay inordinate amounts to give a recommendation as to where your business problems lie, and how you can solve them. The answer is invariably that you need more money.



Beginners' guide to the 6502

Machine code made easy – Peter Turcan tackles the 6502

The 6502 has proved to be a workhorse of a processor. At the centre of the Apple II, Commodore 64, BBC and many other successful microcomputers, the 6502 has developed a momentum which is keeping it alive, even in the days of impressively-powerful processors such as the 68000. One of its main advantages is that it is comparatively easy to program – easier, in our opinion, than the Z-80 – so getting to grips with it does not take too much time.

Every processor has a range of instructions it can interpret – the full range is called the instruction set – and a program written in those instructions is usually called a machine code program. It is possible to add short machine code sections to a Basic program by using PEEK and POKE commands to load the data and instructions into various memory locations.

The reason is simple – speed; Basic is slow, machine code is very fast. Another reason is that a machine code program uses a minimum of memory space, whereas Basic or any other high-level language is comparatively greedy.

Adding machine code to Basic is usually something of an error-prone performance. The instructions sometimes need to be translated into hex before being POKEd into memory, a slow, tedious and unrewarding operation. On some computers such as the BBC, however, that is not necessary as there is a built-in assembler.

An assembler translates what are called mnemonics which a human can understand into numeric codes the processor understands. The process of developing a machine code program is to create a file, usually called a source file, and to use an editor to type-in the instructions in mnemonic form, run an assembler program to create a second file containing those instructions in numeric form, return to the editor to correct any syntax errors, re-run the assembler, and finally run as a program the output from the assembler, usually called a binary or object file.

That basic loop is then repeated many times as faults in the program are found. A further extension of the loop is to develop a large program in several more manageable positions, called modules, assemble them separately but run them all together by using what is called a linker. A reasonable size for any module is 500 lines, so becoming familiar with a linker can be left until you are familiar with the main business of writing machine code programs in assembler.

For anyone considering entering the world of 6502 machine code, I would recommend obtaining a good assembler package and a good reference book on the 6502. The other main piece of documentation needed is a detailed reference guide to the computer memory – a memory map. With those three things machine code programs of great size and complexity can be tackled. Without any one of them you may well fall on the first fence.

Figure 1.
Convention for writing a machine code subroutine

```
SUBRTN  PHA      / save off A,X and Y on the stack
        TXA
        PHA
        TYA
        PHA
        *
        <body of subroutine>
        *
        PLA      / restore A,X and Y
        TAY
        PLA
        TAX
        PLA
        *
        RTS
```

The most fundamental unit of any program is the subroutine, sometimes called a procedure, without which programming would be almost impossible. Before leaping into a program, it is important to work out conventions to handle subroutines and then stay with them through thick and thin. The instruction to call a subroutine in 6502 is JSR <xxxx>, where <xxxx> is the address of that subroutine in memory or, more usefully, a name associated with an address.

JSR is a good example of a mnemonic – it stands for Jump to SubRoutine. When that instruction is done the computer remembers where it finished by placing a return address on a stack. A subroutine is left by using an RTS instruction – Return from SubRoutine – which picks up the return address and processing continues with the next instruction in sequence after the JSR <xxxx>.

A stack is like a machine gun cartridge – the first bullet in is the last out. That method is used so that if one routine calls another, that calls a third, that calls a fourth, and so on, then the RTS instructions will pick up the return addresses in reverse order, so the fourth subroutine will return to the third, the third to the second and finally back to the first.

A processor contains a number of registers – see the computer course in the May issue of *Your Computer* – called the A, X and Y registers. There is also a stack pointer (SP) and program counter (PC) in the processor but if your programs are well-behaved and conform to good conventions, you will not need to concern yourself too much with them. Another register, the processor status (P) contains single bits, called flags, which are set independently to give information on the results of comparisons and the like.

When a subroutine is called it is a good idea to save the

Figure 2.
Adds N1 to N2, leaving the result in N2

```
ADD16   SAVE            / save registers
        CLC             / clear the carry flag
        LDA   N1L       / A register = low byte of first number
        ADC   N2L       / add the low byte of second number
        STA   N2L       / store in low byte of second number
        LDA   N1H       / A register = high byte of first number
        ADC   N2H       / add the high byte of second number
        STA   N2H       / store in high byte of second number
        RESTORE         / restore registers
        RTS
```

A, X and Y registers, in case they contain useful information the calling routine does not want to be corrupted by the subroutine. That gives us our first convention, a subroutine should look like figure one. Whenever the subroutine SUBRTN – that could be any name – is called by a JSR, the registers are saved on the stack before anything is done and restored at the end of the routine before returning.

The five statements which save and restore registers can, in many assemblers, be grouped into what is called a macro. A macro is a definition of any number of instructions and prevents having to type them every time; the assembler program will include them when it encounters the macro. In future examples we will assume there are two macros, SAVE and RESTORE, which save and restore the A, X and Y registers respectively.

You need an assembler, a reference book and a memory map

The 6502 operates on units in byte quantities. There is an add and a subtract instruction but many programs will require additions and subtractions of numbers larger than eight bits. If we wish to add two 16-bit integers, that is done in 6502 instructions by adding each half at a time and detecting whether there is anything which should be carried from one addition to the other. Figure two shows the program.

The program in figure two shows how the carry flag – one of the bits in the P register – is useful. That flag is first set to 0 but the addition of the two low bytes may set it to 1 if that addition overflows. If it is set to 1, the ADC command – ADd with Carry – in the second addition ensures that overflow is taken into account. The other instructions in that subroutine shoud be reasonably obvious; LDA loads the A register with the contents of the memory location specified, STA does the reverse.

Many 6502 programs will contain a large number of fundamental routines which perform 16- and 32-bit addition, subtraction, multiplication and division. Fortunately there is no need to re-invent those wheels by determining all the overflows for yourself; instructions for all those routines will be found in most 6502 programming books from which they can be copied.

Our example addition exposes another problem, that of addressing. The numbers N1 and N2 are passed as parameters to the subroutine by first loading the values into four bytes of memory. Figure three shows how this subroutine would be called to multiply 300 by 125.

In figure three, the instruction LDA £1 loads register A with the number 1. That is called immediate addressing. In figure two the instruction LDA NIL loads the register A not with a number but with the contents of the byte pinpointed by the name NIL. That is called absolute addressing. Immediate and absolute are called addressing modes and there are 13 addressing modes recognised by the 6502 processor.

Addressing modes

Another example is LDA (LOC),Y. That loads the accumulator with the byte which is Y bytes away from the address held by the location LOC in memory. This is an example of indirect indexed addressing; the Y register is used to provide the index and the brackets round LOC mean that the contents of LOC are required and not LOC. The latter form of addressing is called indirection and is very useful in writing machine code programs.

Although addressing is crucial, learning the addressing modes is a little like learning English tenses in grammar; it is often easier to learn by example and ignore the theory. Ploughing through text on indirect, indexed, immediate, absolute, implied and the other modes of addressing is sufficient to send any newcomer running for the Basic Interpreter. Addressing, however, is nothing like so daunting as it looks at first and the initial horror can quickly be overcome by getting on with the job. So learn addressing as you proceed – do not attempt to swallow it all at once.

Another crucial concept surrounds the use of a special 256 bytes of memory called Zero Page. That is the first 256 bytes of RAM, and is special as the first two hex digits of any of those bytes addresses is 00 – hence the name Zero Page. To take advantage of that there are special addressing modes for Zero Page which have the advantage of being faster than the rest, as a shorter address has to be interpreted by the processor.

The address is shorter since the processor knows the first byte of the address is zero and so ignores it. That means that data can be stored and retrieved in zero page faster than in any other area of memory and so that page becomes prime real estate for storing frequently-used variables.

Machine code programming is often easier if the algorithms are written in advance in a well-structured language such as Pascal or possibly Basic. That is because major program flaws are often difficult to find in an assembler listing and also because translating downwards to assembler is straightforward. The statement IF C = D THEN PROC-ONE ELSE PROC-TWO is translated in figure four; note how the high-level language can be used as comments. OUT and P2 are labels used by the unconditional jump instruction JMP and BNE – branch if equal flag not set. Gotos, case statements, FOR loops, WHILE loops and so on can be translated just as easily into assembler.

The few examples should indicate what programming in 6502 might be like. It is true that many more lines of code are needed to translate single lines of a high-level language but with good conventions the extra size should not be too difficult to cope with. Each instruction – there are 56 – usually takes between two- and four-millionths of a second to complete.

Speed of that kind is hopelessly unobtainable in high-level languages and is certainly the main attraction of machine code programming but also there is the feeling that you are really in control of the computer, as machine code instructions are usually easy to understand, perform exact operations and have no side effects.

Figure 3.
Setting-up parameters for the 16-bit addition,

```
/ Data bytes set up in memory
N1L   HEX  00
N1H   HEX  00
N2L   HEX  00
N2H   HEX  00
/     add 300 to 125
      LDA  £1      / A holds 1, not the contents of location
                   1
      STA  N1H     / high byte, so value is 256
      LDA  £44
      STA  N1L     / low byte contains 44 as 44+256 = 300
      LDA  £0
      STA  N2L     / high byte of 125 is 0

      LDA  £125
      STA  N2L     / low byte contains 125
      JSR  ADD16   / performs the addition
```

Figure 4.
Translation of if . . then . .else

```
      LDA C
      CMP D          / IF C = D.......
      BNE P2
      JSR PROC-ONE   / THEN PROC-ONE...
      JMP OUT
P2:   JSR PROC-TWO   / ELSE PROC-TWO
OUT:
```

Forced into submission

The CP/M submit utility allows users both to personalise their systems and to prevent keying repetitive sequences of commands at the keyboard. Ken Alexander introduces the command.

Between the twin extremes of programming in a high-level language like Basic and low-level coding in an MPU native machine language there is a third level of programming – programming in the operating system of a computer. It is more like using a high-level language yet the degree of control it affords the user over the computer system is more akin to that achieved when using machine-code.

CP/M, the most widely-used 8-bit operating system, has a wealth of transient program utilities available to the user, though most people make little use of them. Amstrad, for example, supplies a comprehensive range of CP/M utilities with the PCW and CPC computers yet, with the exception of the disc copy commands, few users will make use of them. At least one of the commands can make using most PCW software, in particular, far easier. That command is SUBMIT; the COM file for this utility is supplied on side two of the PCW system software.

The SUBMIT utility allows a sequence of instructions stored previously within a data file to be executed one by one as if they were entered at the keyboard. To appreciate the significance of this command, consider this example. The PCW computer is to be used in conjunction with a word processor which requires a COM file and associated overlay files to be copied to the PCW RAM disc so that it may operate most efficiently.

Submit is only one of a range of powerful CP/M utilities.

To achieve that without the use of the SUBMIT utility would require the following series of commands to be entered at the keyboard. First, with side two of the computer system software in drive a, the CP/M Peripheral Interface Program would be called with the command PIP. The disc in drive a would then be removed and the word processor work disc inserted in its place. Then the line m:=a:wp.com <return> would be entered, followed by, for example, m:=a:wpmsgs.ovl <return> and m:=a:wpprint.ovl.

Those commands would then be followed by a single return to exit the PIP utility and then the line M:<return> to log on to drive m, then the command WP to run the word processing software. That time-consuming series of commands would have to be entered at the start of every session with the word processor; clearly there must be an easier way. The SUBMIT utility provides that easy way out of the problem.

In the foregoing example, the user would create a file called USEWP. The file type would be set to SUB. Within the file the sequence of commands shown in listing one would be entered. This file could be created with the CP/M ED utility or by Locoscript using the create a simple ASCII file option, having entered the lines of text.

Having created that file on the word processor working disc and copied the PIP.COM and SUBMIT.COM files from side two of the system disc, to load and run the software it is necessary only to type the single command SUBMIT USEWP. The computer will then read in each line of the SUB file and execute each in turn, duplicating the entry of each via the keyboard.

The SUBMIT command has an additional sophistication which operates when the CP/M operating system is first booted. As part of the boot process, CP/M looks for a special file, PROFILE.SUB. If that file is present on the disc which holds the CP/M EMS file, the commands contained in the file will be executed automatically.

To create a boot disc which will allow the WP software to be loaded from a cold start the CP/M operating system software would have to be copied to the WP work disc and the file in which listing one is stored be re-named to PROFILE.SUB. A directory of the WP work disc would thus produce the output shown in listing two.

Many other CP/M commands can be incorporated as part of a SUBMIT file. The SETKEYS.COM and associated KEYS file could, for example, be used as part of the submit process. That would allow the keyboard of the computer to be re-configured so that the keys associated with word processing functions – the cursor control keys, EOL, copy, cut and the like – would generate the appropriate sequence of commands.

The SUBMIT command, if used correctly, can ensure that when the CP/M software is loaded the computer is configured for immediate use. SUBMIT, though, is only one of the commands which can be used to personalise the PCW computer to meet the requirements of a particular user. Browsing through the CP/M section of the PCW manual will reveal many more useful utilities.

Listing 1.

```
PIP
<m:=a:wp.com
<m:=a:wpmsgs.ovl
<m:a:wpprint.ovl
<
<m:
<wp
```

Note that the above could be shortened by use of the CP/M wild card option, by replacing the three lines transferring the WP files to the RAM disc with a single line of the form m:=a:=wp*.*

Listing 2.

```
J14CPM3.EMS
PIP.COM
SUBMIT.COM
PROFILE.SUB
WP.COM
WPMSGS.OVL
WPPRINT.OVL
```

Note that the CP/M software may have a slightly different reference from that shown above but will be the only file to have the EMS extension.

NEXT MONTH

MASTERING INTERRUPTS

Many programs use interrupts to speed their operation. We explain the concept of interrupt-driven software, with some specific examples.

FURTHER READING

Is there life after the *Your Computer Course?* We list some suggestions for additional reading.

AND SO TO Z . . .

. . . With the final part of our glossary.